CPU Resource Management in Personal Computers - A Low Cost Implementation

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

by

SAURABH C SOMAN

to the

DEPARTMENT OF INDUSTRIAL & MANAGEMENT ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

JANUARY, 1995

CERTIFICATE

It is certified that the work contained in the thesis entitled " **CPU Resource Management in Personal Computers - A Low Cost Implementation** ", by SAURABH C SOMAN, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree

(S SADAGOPAN)

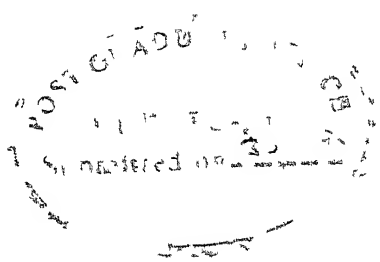
Professor

Industrial & Management Engineering

Indian Institute of Technology

Kanpur - 208016

January, 1995



IME-1995-M-SOM-CPU

ACKNOWLEDGMENT

I take this opportunity to express my sincere and heartfelt thanks to *guruji* Dr S Sadagopan for his invaluable guidance towards academics as well as philosophy of life Thanks are also due to Mrs Sadagopan, who always made feel me at home

I thank my batch mates and friends for their support I can not find words to thank Sqn Ldr J J Wani Sir & Family, who made my stay at IIT-K so memorable

Saurabh C Soman

CONTENTS

LIST OF FIGURES

ABSTRACT

In the present implementation, an attempt has been made to find a low cost solution for efficient computer resource management in Intel 80386 class Personal Computers The thesis describes the hardware and software required to implement a multi-user operating system, which has capability to run multiple DOS processes, using a common keyboard and two independent display devices With minor hardware additions an independent keyboard to control the displays separately can also be implemented

CHAPTER I
INTRODUCTION

1 1 Motivation

In today's World, the micro - processors are getting faster and smarter day by day The Intel 80386 and its later cousins 80486 and Pentium are powerful enough to handle multiple users, performing simple tasks During his summer training the author observed that a large number of 386 class machines were mainly used for text based word processing in the corporate sector The senior executives could afford 386 class machines but the tasks undertaken were utilizing the micro - processor poorly Being expensive and powerful resources they must be utilized more efficiently One way to improve the computer resource utilization is to permit multiple users to simultaneously use the system without the users feeling significant performance degradation This thesis project focuses on a low cost effort towards this purpose

1 2 Multi - User Operating Systems

The desk top Computers which use these processors, usually run Disk Operating System (DOS) mainly from Microsoft MS - DOS can not exploit the power of these computers fully, since it supports only one user

The same system, when running under multi-user operating system, such as UNIX, can support multiple users But a huge software base exits in DOS and the bulk of the executives use only DOS based software Hence DOS support is essential, UNIX can also support DOS with DOS MERGE, by adding an intelligent terminal (low speed) over a

serial connection DOS MERGE and other variations emulate DOS that makes significant performance degradation They support only the partial features of 386 class processors

1 3 A Different Approach

Instead of using an intelligent terminal, to add a user, if only one extra display and keyboard is used, the cost of adding a new user could be halved This also results in much better utilization of CPU Thus two users working in DOS can share the same system's resources, and work simultaneously This must be achieved without any significant performance degradation The users should be able to run any DOS based programs in their individual terminal with no conflicts An implementation that uses low cost hardware and our own software to accomplish this is documented in this thesis

CHAPTER II
OPERATING SYSTEM - THE APPROACH

2 1 The UNIX Solution

The DOS MERGE under UNIX provides DOS support The users are provided with the intelligent terminals These terminals are connected to the Central Processing Unit (CPU), through a serial link The operating System, UNIX, then handles all the I/O for the new user A DOS task which directly talks with the hardware, runs very slowly, since the operating system has to emulate the hardware This excess software overhead, kills the processor time

2 2 Our Approach

Installing only a display and a keyboard on the system I/O channel will improve the performance The interface between the new user, and the CPU is always parallel, since the display card and keyboard controller are installed on the I/O channel, which has a parallel data and address bus The separate display card, residing on the I/O channel directly displays the data written in the display memory DOS MERGE must convert this data, directed towards the display memory, into serial format to send it across to the intelligent terminal Hence the user will not face any delay in display of data, because of the parallel interface The tasks have separate hardware support for display and keyboard, which is compatible with the PC platform This relieves the operating system of emulating this hardware This allows the operating system, to devote more time to the task

A new Core Operating System is required for Resource Management This operating system will resolve the conflicts in sharing the resources, such as disk, diskettes and other add on cards

CHAPTER III
ARCHITECTURE OVERVIEW

The PC - 386 mother board has the following major components which are important from the project point of view

1) The Processor (Intel's 80386),

2) Programmable Interval Timer,

3) Programmable Interrupt Controller (PIC),

4) Direct Memory Access Controller,

5) Keyboard Controller,

6) Memory Mapping & I/O Mapping,

3 1 The Processor [1] [4]

The mother board is provided with Intel's 80386 processor It has 32 - bit internal architecture It can run under different operating modes such as Real Mode, Protected Mode and Virtual 8086 Mode When the processor restarts after a master reset, it works under the real mode And afterwards, it could be switched to protected mode and then to V86 Mode

The 80386 is fully compatible with the 80X86 family of processors The memory capacity of the processor is 4 GB (2 ^ 32) which could be addressed as a flat 4GB chunk or in a segmented fashion It can also access 64 Tera bytes (2 ^ 42) of Virtual Memory using the Page On Demand method The on - chip Memory Management Unit (MMU) can be used to implement the virtual memory very effectively Thus a secondary storage unit could be efficiently used as Virtual Memory

The Input / Output (I/O) address space is 64K It also has a separate I/O address space with a separate protection mechanism for it In protected mode the processor can handle multiple tasks, for which it has in - built memory protection and I/O protection

The low cost version of the 80386 is 80386SX which has 16 - bit external data bus and 24 - bit address bus i e a total address space of 16 Mb, but the internal architecture is similar to 80386DX or 80386 So all the 32 - bit instructions which work on the 80386DX can also work on the 80386SX The MS - DOS in general works in the Real Mode and does not take full advantage of the multitasking facilities of the 80386

3 2 Programmable Interval Timer [2] [3]

The CPU has a system clock for timing Many other basic hardware and software functions occur at regular intervals based on a preset clock frequency

The Timer has three output channels It has it's own crystal with output frequency of 1 1 93 18 MHz The output of the timer can be programmed

All the three output channels have their own dedicated functions

3 2 1 Channel 0

It is the system clock - tick timer When the computer is cold booted it always programs a count of 0000h in the timer When the count passes through 0000h, because of decrementing, a tick is generated on this channel The timer has a 16 - bit count thus it giving 18 2 1 e (1193180 / 655360) ticks per second This output is given to Interrupt ReQuest 0 (IRQ 0) of the Programmable Interrupt Controller which in turn invokes a BIOS routine This BIOS routine each time upgrades the Day - Date - Time of the system, then it also commands the disk drive controller to turn off the motor, if it is on

3 2 2 Channel 1

This channel is used to refresh the Dynamic RAM on the board The DRAMs are required to be refreshed at a fixed interval of time This channel output invokes a DMA channel which in turn refreshes the DRAMs

3 2 3 Channel 2

This channel is used to control the computer speaker The frequency of the timer determines the frequency of the sound emitted by the loudspeaker

3 3 Programmable Interrupt Controller [2] [3]

In a PC, one of the CPU's essential tasks is to respond to hardware interrupts A hardware interrupt is generated by a component of the system This interrupt is routed to

the processor through a Programmable Interrupt Controller such as Intel's 8259 cascaded together to have 15 interrupt levels The I/O mapping is at 20h - 21h for controller # 1 and A0h - A1h for controller # 2 The use of each IRQ line is as shown in the figure 3 1

CTRL # 1	CTRL # 2	Assignment
IRQ 0		Timer Output 0
IRQ 1		Keyboard O/P Buffer Full
IRQ 2		Interrupt from CTRL # 2
	IRQ 8	Real Time Clock Intr
	IRQ 9	Reserved
	IRQ 10	Reserved
	IRQ 11	Reserved
	IRQ 12	Reserved
	IRQ 13	Co-processor
	IRQ 14	Fixed Disk Controller
	IRQ 15	Reserved
IRQ 3		Serial Port 2
IRQ 4		Serial Port 1
IRQ 5		Parallel Port 2
IRQ 6		Diskette Controller
IRQ 7		Parallel Port 1

Figure 3 1 IRQ Assignments

The first controller is given the interrupt mapping of 08h to 0Fh and the second controller is mapped from 70h to 77h i e whenever the system timer gives a tick, it requests the processor attention by sending an interrupt request on IRQ 0 line The IRQ 0 line has the highest priority and the 8259 gives an interrupt to the processor and asks the processor to process INT 08h The processor then executes the code for INT 08h known as interrupt handler The code in the interrupt handler carries out the appropriate hardware specific activity, such as incrementing time of day counter Similarly an interrupt from the hard disk will be processed by INT 76h since the Interrupt Mapping for controller # 2 is from 70h to 77h

All the functions such as priority resolution, Interrupt Masking are controlled by software, and could be changed for other suitable use

3 4 Direct Memory Access Controller [2] [6]

Direct Memory Access (DMA) is a technique by which some peripheral components can transfer data to and from the computer memory without passing the data through the processor DMA is handled by a chip such as Intel's 8237 In a IBM - PC there are two 8237 s cascaded together The use of individual channel is as shown in the figure 3 2

CTRL # 1	CTRL # 2
Ch 0 - Spare	Ch 4 - Cascade for CTRL # 1
Ch 1 - SDLC	Ch 5 - Spare
Ch 2 - Diskette Controller	Ch 6 - Spare
Ch 3 - Spare	Ch 7 - Spare

Figure 3 2 DMA Channel Assignments

The DMA controller (8237) has a 16 - bit count and an address pointer But when used with DMA Page Register it can address more than 1 Mb of memory, which is the upper limit for the processor operating in the Real Mode Thus when 8 - bit page registers are used the DMA controller can access a maximum of 16 Mb of memory The channels 0 to 3 can transfer data in blocks of 64 KB or less and channels 5 to 7 can transfer data between 16 - bit I/O adapters and 16 - bit memory locations in the blocks of 128 KB These channels 5 to 7 cannot transfer data on odd byte boundaries

The DMA is used for DRAM refresh The DMA channel 2 is used for data transfer from Diskette to memory This helps in fast transfer of data to and from the diskette The DMA could be programmed only when it is in slave mode When in Bus Master mode it has full independent control of the Address Bus, Data Bus and Control Bus

3 5 Keyboard Controller [2]

The Keyboard Controller is a single - chip microcontroller such as Intel's 8042, which has on chip program to support the IBM PC - AT Keyboard serial Interface The Keyboard controller receives serial data from the keyboard, converts it to parallel form and checks the parity of the data This data is in the form of scan codes It translates these scan codes to ASCII and then presents them in the output buffer located at I/O 60h The status register contains bits that indicate if an error was detected while receiving the data Data may be sent to the keyboard by writing in the input buffer of the keyboard controller The controller then transmit the data in serial form with an odd parity The interface between the Main Processor and Keyboard controller is as shown in figure 3 3

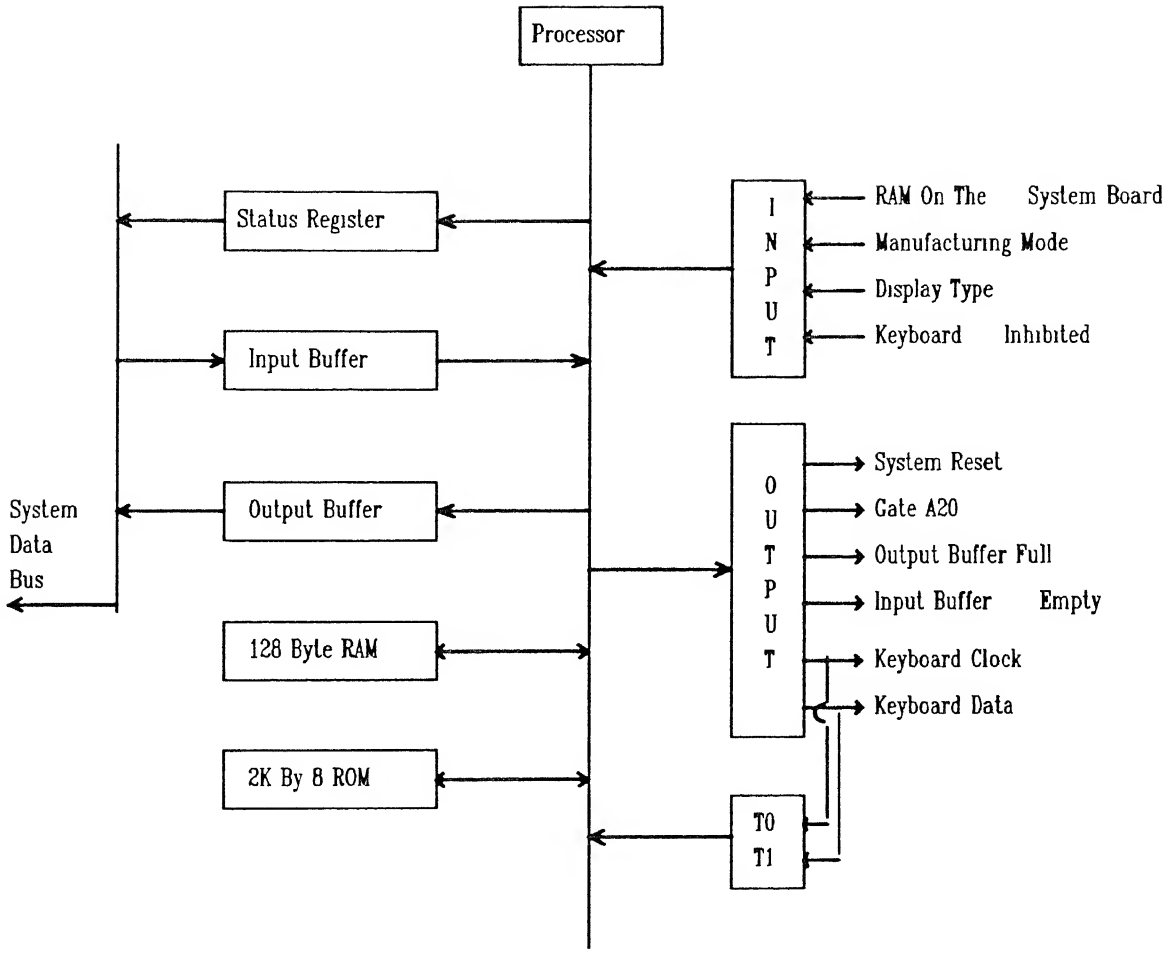


Figure 3 3 Keyboard Controller

Apart from the communication with keyboard the Controller also perform some other functions It provides the status information about the 2nd 256KB of system board RAM, the primary display when attached to the Color Graphics Adapter (CGA) or Monochrome Display Adapter (MDA)

It also controls the Non Maskable Interrupt (NMI) Whenever NMI goes low the system is reset It can mask the A20 bit of the address bus thus limiting access to a maximum of 1MB RAM This A20 bit has to released from the keyboard controller before accessing the 1MB onwards RAM

3 6 Memory Mapping & I/O Mapping [5] [2] [3]

The memory available on the system may vary from 640KB to 16MB The processor has an address space of 1MB in real mode

Some of the layout of the memory is a consequence of the design of the earlier processors such as 8088 The processor always uses the first 1024 bytes as Interrupt Vector Table (IVT) And the ROM is always located at the top This is because after a master reset, the processor starts executing the instructions from the address FFFF0 h

The 640KB limit of memory is a DOS limit and the other area is used for Video RAM buffers, installable ROM & BIOS ROM The RAM which is present beyond 1MB is known as extended RAM and could be accessed by using the DMA or by initializing the protected mode of the processor

The I/O address space is different from memory address space The I/O address space is from 0000h to FFFFh i e 640KB This address space is accessed using IN and OUT instructions The I/O address of 000h to 0FF h are used for system board I/O and addresses 100h to 3FFh are available on the I/O channel which is enough for most of the applications It avoids the decoding circuit required for all the other address lines on the I/O channel

3 7 CMOS RAM [2]

This is a special memory that is maintained by a battery so that it retains information even when the computer is turned off This permanent memory is used to maintain a real time clock and other system parameters such as fixed disk drive types and the memory size

A total 64 Byte of memory is available which is accessed through two ports at addresses 70h and 71h To read or write a byte into the memory requires the byte number ranging from 0h to 3Fh to be written at port 70h and the data byte is read or written at port 71h

3 8 Video Subsystem [3]

The video subsystem is responsible for producing the image that appears on the screen This system may be built on the system board or it may be present as a plug-in card The interface with the system remains the same In the PC I/O map, addresses 3B0-3BFh are used for CGA, EGA or VGA

This system also has memory which is mapped in the memory map after A0000h till BFFFFh Not all this memory is present but part of it is always used

CHAPTER IV
THE 80386 ARCHITECTURE

The 80386 micro - architecture has the functional groups as shown in the figure 4 1

1) Bus Interface Unit,

2) Instruction prefetch Unit ,

3) Instruction Decode Unit ,

4) Execution Unit,

5) Segmentation Unit,

6) Paging Unit,

4 1 Bus Interface Unit (BIU) [1]

The BIU is the 80386 ' s gateway to the external world Any other unit which needs to read or write the data asks the BIU to perform the operation The BIU is presented with the address and data and is asked to place it on the bus The BIU deals with the physical addresses only Hence operand addresses must first pass through the segmentation unit and the paging unit, if necessary

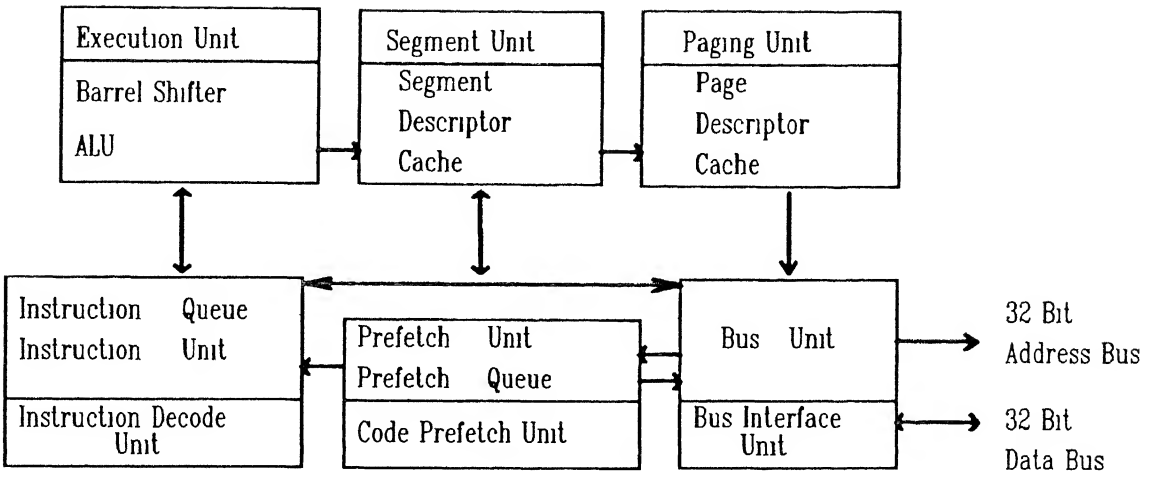


Figure 4 1 80386 Micro Architecture

4 2 Instruction Prefetch Unit [1]

The Instruction Prefetch Unit has a 16 byte queue and it always asks the BIU to fetch the next 32 - bits piece of memory whenever the queue is not full The prefetch unit is notified whenever the execution unit process a CALL, JMP or an interrupt so that it can flush the queue and start fetching from the new address, thus preventing the execution unit from receiving out of sequence instructions

4 3 Instruction Decode Unit [1]

The Instruction Decode Unit fetches one instruction at a time from the queue Then it reformats the op code into an internal instruction format and places the decoded instruction into the instruction queue which is three operations deep

4 4 Execution Unit [1]

The Execution Unit is that part of the CPU which performs the computations It performs any shifts, additions, multiplications and so on, that are necessary to accomplish an instruction The register set is contained inside this unit The unit also contains a logic component called the Barrel Shifter, which can perform multiple bit shifts in a single clock cycle The execution unit also tells the BIU when it has the data that needs to be sent to the memory or I/O bus

4 5 Segmentation Unit [1] [4]

The segmentation unit translates the segmented address into linear address This unit contains a cache that holds descriptor table information for each of the six segments

4 6 Paging Unit [1] [4]

The Paging unit takes the linear addresses from the segmentation unit and translates them to physical addresses When paging is disabled, the linear address becomes the physical address, but when it is enabled, the linear address space is divided into 4096 byte blocks called pages Each page can be mapped to an entirely different physical address The 80386 uses a page table to translate each linear address to a physical address The paging unit contains an associated cache called the Translation Look - Aside Buffer (TLB) which contains the entries for the 32 most recently used pages

4 7 Protected Mode [1]

The Processor starts running in Real Mode after a Master Reset, but after that, it could be switched to Protected Mode Protected Mode allows multiple applications to run concurrently but isolates them from one another so that failures in one application do not affect any other application

4 7 1 Selectors [1]

The central feature of the protection mechanism is the Selector Rather than directly accessing any part of the system, a program deals with a selector, which grants access to a

system object after checking for object's location, size and type and also any restrictions on its use

4 7 2 Descriptors [1]

Each Selector has an associated Descriptor which contains the detailed information about the system object such as Memory Segments, tables which support the protection mechanism such as GDT and LDT, the processor state (TSS) and access control objects such as Gates The descriptor has all the information about the kind of object, location, size and privilege level

4 7 3 Privilege Level [1] [4]

The Protection Mechanism supports four levels of increasing Privilege, numbered 3, 2, 1 and 0 Privilege level 0 is the most privileged level The most reliable and crash resistant code should be allowed to run at the privilege level 0 And applications that might fail and compromise the integrity of the system should be run at the least privileged level of 3

4 7 4 Gates [1]

Whenever there is a need of communication between two different levels of privileges it is to be carried out using Gates

A Gate is a system object i e it has its own descriptor, that points to a procedure in a code segment, but the gate has a privilege level separate from that of the code segment

The Protection Mechanism supports four types of Gates, CALL gates, INTERRUPT gates, TRAP gates and TASK gates

The CALL gates are invoked by standard subroutine calls

The INTERRUPT and TRAP gates are invoked by INT instructions or by hardware interrupts

The TASK gates are invoked by JMP, CALL, INT, or hardware interrupts

4 7 5 Task State Segments [1] [4]

Whenever there is a change in TASK the processor stores the state of processor in a special segment called Task State Segment (TSS) It contains a copy of all the registers that must be saved to preserve the state of a task It also contains values associated with task but those are not stored in CPU registers

4 7 6 Descriptor Tables [1] [4] [2]

All the descriptors are grouped into Descriptor tables There are three types of Descriptor tables - Global Descriptor Table (GDT), Local Descriptor Table (LDT), and Interrupt Descriptor Table (IDT) The GDT contains all types of descriptors but the IDT contains descriptors related only to hardware and software interrupts

4 8 Interrupt [1] [4] [2]

Interrupt is a term that refers to a variety of similar control transfers The specific items implied by this term are True Interrupts (hardware interrupts) and Exceptions which are further divided into Traps, Faults and Aborts

4 8 1 Interrupts [1]

True interrupts are caused by hardware signals that originate outside the CPU Two pins NMI and INTR are used to give the signal The NMI always invokes the routine associated with interrupt vector or IDT entry 2

4 8 2 Traps [1]

These are conditions that the processor regards as errors and detects after the execution of a software instruction All software interrupts are handled as traps

4 8 3 Faults [1]

When the execution unit detects an error during the processing of an instruction, such as an instruction operand, which is stored in a page frame marked not - present, a fault occurs A specific number entry in IDT is associated with each fault condition

4 8 4 Aborts [1]

When an error is so severe that some context is lost, the resulting condition invokes an abort Usually, the instruction causing an abort can not be restarted

The figure 4 2 lists all the exceptions handled by processor

Interrupt #	Description
0	Divide Error
1	Debugger Interrupt
2	Non - maskable Interrupt
3	Breakpoint
4	Interrupt on Overflow
5	Array Boundry Violation
6	Invalid Opcode
7	Co - processor not available
8	Double Fault
9	Co - proc segment overrun
10	Invalid TSS
11	Segment not Present
12	Stack Exception
13	General Protection Fault
14	Page Fault
15	Reserved
16	Co - processor error
17	Alignment Check
18 - 31	Reserved
32 - 255	System Dependent

Figure 4 2 80386 / 486 Exceptions

4 9 Paging [1] [4]

The processor can handle a maximum 64 Tera bytes of memory using virtual memory Paging is used to implement virtual memory based on fixed size blocks called Pages Paging translates virtual address into physical address In 80386 each page is 4096 bytes Since the

total physical address space is of 4 GB, more than 1 million page entries are required to implement the whole memory The paging is implemented as a two level page table The high order 10 bits are used as an index to a page directory and then the next 10 bits are used to select an entry from a page table When paging is invoked, only one register contains a physical address pointing to the base of page directory When a task asks for an address, which is present in a page frame marked not-present, then an exception is generated Then the page is swapped in and then the faulting instruction is restarted Each page could be implemented as byte granular or page granular When the page granularity is used all the address space of 4 GB is implemented as a single chunk

4 10 Virtual 8086 Mode [1] [4]

Just as virtual memory allows the processor to create the impression of memory that really isn't there, Virtual 8086 mode allows the 80386 to create the illusion of multiple 8086 processors In V86 the task does not use selectors Base addresses are generated by multiplying the segment address by 16 The only difference between real mode addresses and V86 addresses is, the real mode addresses are physical addresses while the V86 addresses are virtual addresses which could be mapped using paging

In Protected Mode, the I/O privilege level (IOPL) determines whether a procedure can perform I/O instructions In V86 mode, IOPL protects the interrupt flag (IF) and I/O port protection is performed through the I/O permission bits in the TSS The instruction below generate a General Protection Fault (Interrupt 13)

CLI POPF INT PUSHF IRET STI LOCK

I/O instructions are not IOPL sensitive in V86 mode When the I/O instructions are executed without the I/O privilege for the task, the following instructions also return General Protection Fault

IN INS OUT OUTS

But these instructions could be restarted

Because V86 mode is a part of the Protected Mode environment, interrupts are handled through the standard protected mode IDT When IOPL is less than 3, a software interrupt generates a general protection fault So the interrupt handler for General Protection Fault (Interrupt 13) should be able to emulate these instructions for the V86 task to run properly

CHAPTER V
DESIGN AND IMPLEMENTATION

5 1 Protected Mode Booting Conflicts

The Goal is to run two DOS tasks in V86 mode The DOS is loaded after some checks When the system is turned, on the Power On Self Test (POST) is carried out, the boot sector is loaded and in due course the DOS starts running But after starting the protected mode if POST routines are executed, then there will be direct conflicts The POST itself will try to initiate the protected mode to check the extended memory, and afterwards to return to the real mode, a system - reset is given [2] Thus by following the usual procedure it is not possible to create one more DOS image in the system

A copy of the DOS image is run instead of booting the system A snap shot of the DOS image present in the system is taken A memory buffer of 640k will be required to store a snap shot of the DOS image But if the transient part of the DOS is loaded from the disk, then a buffer of 128K is sufficient to store all the necessary DOS image characteristics Thus, when the present process running in this DOS image, is terminated by the DOS INT 21h function Terminate Process, the transient part will be loaded from disk When DOS loads and executes a process, all the available memory is allocated to the new process It is the responsibility of the process to release the excess memory back to DOS Once this excess memory is released, a buffer of 128KB is asked from the DOS in the upper memory A snap shot of the DOS image is stored in this buffer This ensures that when the memory contents of the low memory are written over, this memory buffer has a safe copy of the low memory All the required characteristics of the DOS image are safe at known memory

location in the buffer Thus, all the low memory of 128KB is available for building the new operating system image for the protected mode operations

5 2 Initialization

Protected Mode initialization requires an image of the Global Descriptor Table (GDT), an Interrupt Descriptor Table (IDT), and a Task State Segment (TSS) of the first process

5 2 1 Global Descriptor Tables

A GDT is prepared at the 00000h This destroys the IVT of the real mode, hence the Interrupt Flag is cleared before creating the GDT The GDT(0) is unused because a selector of 0h is treated as a special case called as the NULL pointer [1] GDT (1) points to the GDT as a writeable data segment, allowing the new core operating system to add, delete and change descriptors as needed GDT (2) points to the IDT for the same reasons GDT (3) defines the Task State Segment (TSS) for the startup task, GDT (4) defines the tasks data segment and GDT (5) the code segment All the above entries are essential The other entries are built when those are required while defining the V86 tasks

5 2 2 Interrupt Descriptor Table

The IDT entries are as per the exception assignments pointing to specific routines in the code of the operating system

5 2 3 Task State Segment

The TSS for the process is initialized with access to all the I/O addresses The TSS for the two V86 tasks are also prepared The V86 task always has a Current Privilege Level (CPL) of 3 [1] This TSS also has a stack area for the routines which work at CPL of 0, when invoked by an exception

5 3 Access Rights

The access to some of the I/O space is kept limited only to CPL 0 routines, and thus any V86 task I/O instructions, which try to address the protected mode addresses generate a General Protection Fault

5 3 1 Keyboard Controller

Both the V86 tasks have no direct access to the Keyboard Controller Any routine which has access to the controller can give Master - Reset to the system and can also latch the 20 th address bit This can cause a system crash So the access is granted by the Core Operating System only after making sure that the instructions given to the controller won't cause a system crash

5 3 2 Direct Memory Access Controller

One of the other protected I/O space is of the DMA page registers The data transfer is carried out between the diskette and the memory, by using the DMA The DMA can access all the memory space, from 0 to 16 Mb The DMA uses the page registers to access the

memory and always deals with the physical addresses, with no regards to the paging system followed by the processor When a V86 task, with logical address space of 0 to 1 Mb, but mapped in the physical memory after the 1 Mb mark can cause fatal errors in the operation The task can ask for data from the diskette, and pass a address of a buffer within the 0 to 1 Mb logical address space to the DMA Physically the task is present at address space of 1 Mb to 2 Mb The DMA will start transferring data, to the logical address, assuming it as the physical address This can destroy the memory contents of an address space not belonging to that task This can crash the system To avoid this the core operating system checks the addresses passed to the DMA page registers The addresses are passed only after changing them according to the paging of the processor This ensures that the data reaches the V86 task, which had requested it

The checking of the I/O instructions is also carried out for the Interrupt Controllers, Timer and the Math co - processor

5 3 3 Display Adapter

The address space, for the EGA display card is protected for task # 1 The other task has the direct access The trapped I/O instructions of one task are mapped on to another I/O address space, which is available on the system I/O bus but still unused The Display Card of task # 1 also has to be mapped in the same I/O address space The mapping of the this card is done by hardware mapping techniques

5 4 Creating Multiple DOS Tasks

Both the TSS of the V86 tasks, point to a real mode code which uses the DOS INT 21h function to terminate a process Thus when the V86 tasks are initiated, the individual tasks will get terminated, and will load the transient portion of DOS from the disk Thus both the V86 tasks, will have their own DOS copies running independently

5 5 Initial Page Tables

The page tables are created for the primary Core The page directory shows that only one page is present All the rest of the pages are marked as not-present The page table shows that only first 2 Mb memory is present, and the rest is marked as not-present The memory marked present is mapped in such a fashion, that the physical to logical mapping does not change the addresses This is suitable for the initial simple model

The logical addresses of the GDT base and the IDT base are loaded into the GDT base register and IDT base register respectively

5 6 Gaining Access to Extended Memory

In the PC - Architecture, the A20 address bit is latched to 0 [2] Thus, the access to the memory in the address space 1 to 2 Mb is not possible unless the A20 bit is released This address bit is gated through a gate, controlled by the keyboard-controller The keyboard - controller is instructed to release the A20 bit gate This allows the access to the memory beyond the 1 Mb

5 7 Re - mapping the Interrupt Controller

The PC - 386 has 15 interrupt levels Out of which the first eight interrupt requests are mapped at 08h to 0Fh by DOS The remaining are mapped at 70h to 77h But the same interrupt space of 08h to 0Fh is used by the processor, for the exceptions in Protected Mode This conflict of interrupts is solved by mapping the true interrupts at locations 20h onwards These interrupts are taken by the core operating system, and then are passed to the V86 tasks, mapped properly according to the DOS map of interrupts The mapping of these interrupts is carried out without changing the Interrupt Mask in the Interrupt Controller

5 8 Enabling the Protected Mode

The protected Mode is started with paging enabled, by writing the Control Register 3 (CR 3) of the processor In the protected mode the selectors are used in the segment registers To flush the Real Mode segment register values a FAR jump is required Each individual task has a paging directory and a page table Though each V86 task deals with logical address space of 0 to 1 Mb, there is no conflict in the actual physical locations The DOS uses 640 KB memory The remaining of the memory is used for other purposes, and part of it is ROM Hence both the V86 tasks can share this memory without conflict This 384 KB memory out of the 2 Mb memory is available for the core operating system It is also used for storing the initial DOS process characteristics A copy of this is used whenever a V86 DOS process asks for a new copy of DOS Transfer the GDT, IDT and the rest of the operating system to this 384 KB memory, located in the address space of 1A0000h to 200000h as shown in figure 5 1

5 8 1 Preparing For The V86 Tasks

A descriptor is created to access the memory buffer, which has a copy of the IVT, the various data areas, and the resident portion of DOS This buffer is copied to a new memory buffer in the 384 KB memory as shown in figure 5 1 Also a copy of this buffer is placed at the lower most addresses in both the V86 tasks Hence both the tasks have all the necessary information to run independent DOS processes

5 8 2 Executing The V86 DOS tasks

First the interrupts are enabled and then V86 task # 1 is executed for the first time The IP and CS in the TSS of the task points to the DOS INT 21h function to, terminate a process As this process in the V86 DOS environment is terminated, the transient part of the DOS is reloaded in the RAM and the usual DOS prompt is given for task # 1 on screen # 1 When the other V86 task # 2 is started, it also passes through the same phases and gives a DOS prompt of task # 2 on screen # 2

5 9 The Real Mode & V86 Mode Operational Differences

The tasks are now operating in the V86 mode, instead of the Real Mode There are differences between these two modes All the instructions dealing with the Interrupt Flag (IF) generate a General Protection Fault (GPF) The INT instruction also generates a GPF Any I/O instruction referring to a protected I/O address for that task generates a GPF All these instructions are to be checked and emulated for the V86 tasks When the above instructions generate a GPF, exception handler # 13 gains control from the tasks This

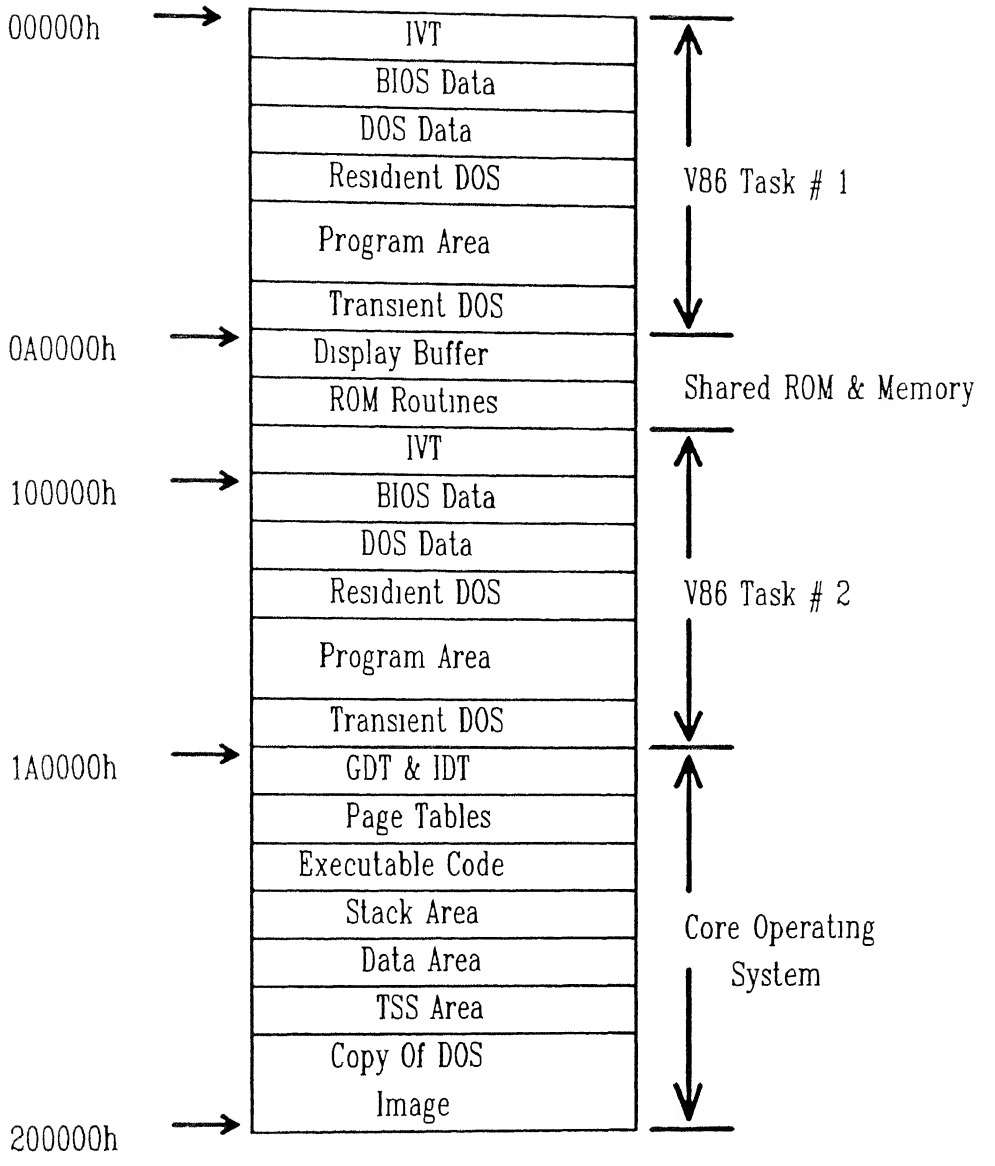


Figure 5 1 Core Operating System Memory Map

handler has a CPL of 0 The IP in the TSS points to the faulting instruction and depending on the instruction further action is carried out

5 9 1 INT

All the software interrupts are two byte long, except INTO, which is only one byte long The Stack Segment (SS) and Stack Pointer (SP) of the tasks are read from the TSS The SP and IP are updated, and then stored along with the other registers, at the memory location pointed by the SS SP in the memory area of the task

Then the Interrupt Vector is read from the task image The task is again restarted from this new vector position Thus the INT instruction is emulated for the V86 DOS task

5 9 2 IRET

This is a one byte instruction which pops the flags, the CS, and the IP from the task Stack Since the Flags are popped from the task image, this instruction can modify the IF of the processor So while emulating this instruction, all the care is taken so that the IF of system is not modified, and only the IF status of the task is updated This local IF status is then used, for the events and instructions which need to know the IF status

5 9 3 CLI & STI

Both these instructions are one byte long, and deal with the IF CLI clears the IF flag of the system and STI sets the IF Both these instructions are emulated in such a way, that the local copy of IF is updated

5 9 4 PUSHF & POPF

Both these instructions are one byte long and deal with the IF Always the next instruction after these instructions, is restarted The change in the IF status is limited only for the local copy The other register values are popped from the task image and passed to the task through the TSS

5 9 5 IN & OUT

These instructions generate a GPF, only if the I/O addresses addressed by these instructions, are protected by the I/O permission Bit Map in the TSS

All the IN instructions are executed for the faulting task The result is passed to the task through the task TSS The task is then restarted from the next instruction

The OUT instructions are checked more rigorously All the implications of the instruction are checked, and the operand is modified to make it safe Only then the instruction is executed, for the faulting task The task is then restarted from the next instruction The I/O instructions of task # 1, addressing to the EGA address space (3C0h - 3DFh) are trapped, generating a GPF All these instructions are executed for task # 1, by the GPF handler,

mapped onto a free address space (2C0h - 2DFh) The display adapter of task # 1 is mapped onto this address space using hardware mapping techniques

5.10 Interrupt Handling

The hardware or true interrupts are always received by the Core Operating System All the precaution is taken so that, the IF is not cleared by any of the tasks All the interrupts are mapped properly according to the DOS interrupt map and then passed to the tasks The tasks run on time sharing basis The task switching is carried out on the basis of the system timer The system timer gives a fixed number of ticks per second Half of the time is given to task # 1 and the remaining is given to task # 2

5.11 I/O Channel Features [2]

The Industry Standard Architecture (ISA) I/O bus on the PC mother board supports

* I O address space from 100h to 3FFh,
* 24 bit memory addresses providing an address space of 16 Mb,
* Interrupts
* DMA channels,
* Open bus structure, allowing multiple micro - processors to share the system's resources, including memory
* Refresh of system memory from channel Micro - processor,

The connector consists of one 62 pin and one 36 pin edge connectors All the signals on the bus are I IT compatible The different signals on the bus are as explained below

5.11.1 SA0 - SA19 (I/O) [2]

This is the system address bus. These 20 address lines can access a maximum of 1 Mb of memory These lines are gated on the system bus when " BALE " is high, and latched on the falling edge of " BALE ". These signals are available for whole cycle.

5.11.2 LA17 - LA23 (I/O) [2]

These signals also form a part of the Address Bus. But unlike SA0 - SA19, these signals are not latched So these signals are not valid for the whole cycle, and are valid only when BALE" is high These signals give the system up to 16 Mb of addressability.

11.3 SD0 - SD15 (I/O) [2]

This is the data bus available on the I/O channel.

11.4 BALE (0) [2]

' Bus Address Latch Enable " is used to latch valid addresses on the address bus. "BALE" " AEN " are forced high during the DMA cycle.

11.5 IOR (I/O) [2]

I/O Read " instructs an I/O device to drive its data onto the data bus. This signal is ive Low.

5 11 6 IOW (I/O) [2]

" I/O Write " instructs an I/O device to store the data present on the data bus This signal is active Low

5 11 7 SMEMR (O) & MEMR (I/O) [2]

These signals instruct the memory devices to drive data onto the data bus " System MEMory Read " is active only when memory decode is within the Low 1 Mb memory space " MEMory Read " is active on all memory read cycles

5 11 8 SMEMW (O) & MEMW (O) [2]

These signals instruct the memory devices to store data present on the data bus " System MEMory Write " is active only when the memory decode is within the Low 1 Mb memory space " MEMory Write " is active on all memory write cycles

5 11 9 AEN (O) [2]

" Address ENable " is used to degate the micro - processor and other devices from the I/O channel and to allow DMA transfers to take place When this line is active, the DMA controller has control of the Address Bus, the Data Bus, and the Control Bus

5 11 10 Refresh (I/O) [2]

This signal is used to indicate a refresh cycle and can be driven by a micro - processor on the I/O channel

5 12 Display Card Memory Mapping

The Display Card is mapped at I/O addresses 3C0h - 3DFh And the memory is mapped at 0A0000h to 0BC000h The display card uses the signals SMEMR and SMEMW So the card responds only to the addresses lying in the Low 1 Mb For the V86 task # 1, the Logical addresses of 0A0000h to 0BC000h are physically mapped at location 8A0000h to 8BC000h The signals MEMR and MEMW, are gated with the memory decode of 8A0000h through 8BC000h, as shown in the circuit diagram 1 Thus two new signals Video MEMory Read (VMEMR) and Video MEMory Write (VMEMW) are derived These two signals are used instead of the SMEMR and SMEMW respectively Thus the memory on the video card responds to addresses of 8A0000h to 8BC000h

5.13 Display Card I/O Mapping

The I/O space of the V86 task # 1 is mapped at location 2C0h to 2DFh, instead of 3C0h to 3DFh So the I/O mapping of the video card is carried out by changing the status of the address bit SA8, of the address bus Whenever the address on the address bus is not in the address space of video display RAM, this bit SA8 is inverted Thus the address 0000 0010 110X XXXX b i e 2C0h to 2DFh will be 0000 0011 110X XXXX b i e 3C0h to 3DFh, which is the usual I/O address space of the Display Card Hence the display card will respond as usual, for the I/O address of 02C0h to 02DFh In this way the display card of

V86 task # 1 will not have a conflict with V86 task # 2 Thus both the displays will respond to their own tasks

CHAPTER VI
CONCLUSIONS

6 1 Results

In the current implementation the capacities of the micro - processor 80386 are used better, when working in the protected mode In the protected mode, the processor has the ability to run real mode processes in the V86 mode Thus multiple V86 mode processes are executed simultaneously Each of the two processes is having a copy of DOS, and is totally independent of each other No understanding is required, on the part of the processes, for sharing the system resources

All the known vital functions of the system are well guarded, in our implementation The Interrupt Flag (IF), the System Reset, the Interrupt Controller, and such vital functions are available for the individual tasks, but the effect of the operation is limited only for that particular task In case a V86 task tries to execute an instruction which can't be emulated, such as invoking the protected mode, the task is terminated and a new copy of DOS is issued to it So if one task crashes, the other task is safe and running

The tasks share the resources such as the disk and diskette Both the tasks have their own copies of the File Allocation Table (FAT) And if one task updates the FAT, and writes to the disk, the other task is also informed of the same Thus data written by one task is not over written by the other The change of task is avoided, when one of the tasks has asked for the disk access This is to ensure that, when one task asks for data, the data reaches the task which had asked for it

CHAPTER VI
CONCLUSIONS

6 1 Results

In the current implementation the capacities of the micro - processor 80386 are used better, when working in the protected mode In the protected mode, the processor has the ability to run real mode processes in the V86 mode Thus multiple V86 mode processes are executed simultaneously Each of the two processes is having a copy of DOS, and is totally independent of each other No understanding is required, on the part of the processes, for sharing the system resources

All the known vital functions of the system are well guarded, in our implementation The Interrupt Flag (IF), the System Reset, the Interrupt Controller, and such vital functions are available for the individual tasks, but the effect of the operation is limited only for that particular task In case a V86 task tries to execute an instruction which can't be emulated, such as invoking the protected mode, the task is terminated and a new copy of DOS is issued to it So if one task crashes, the other task is safe and running

The tasks share the resources such as the disk and diskette Both the tasks have their own copies of the File Allocation Table (FAT) And if one task updates the FAT, and writes to the disk, the other task is also informed of the same Thus data written by one task is not over written by the other The change of task is avoided, when one of the tasks has asked for the disk access This is to ensure that, when one task asks for data, the data reaches the task which had asked for it

The displays of the individual tasks are separate Both the displays are independent of each other One display can be working in the graphics mode, and the other in the text mode, without any problem

The minimum memory requirement is of 2 Mb Both the tasks feel that they have 1 Mb of memory But part of the memory is shared, thus providing the space, where the Core Operating System can install itself

6 2 Limitations

In our implementation when two computation intensive jobs are in progress, the processor utilization is not improved Also when one of the tasks is computation intensive, and other non computation intensive, the processor utilization falls, as compared to the real mode operation

Though all the known ways, which can crash the system, are guarded, all the pitfalls are not identifiable Hence during operation the system may pose problem

6 3 Conclusions

The utilization of processor is improved, at the cost of one display card, display, keyboard controller, and keyboard One extra user can be added to the system When the jobs are not computation intensive, this system can improve the overall utility of the available resources

6 4 Scope For Future Work

At present, the system shares the keyboard Each task could be provided with a separate keyboard The procedure is exactly similar to that of providing multiple displays, trap the I/O space of the keyboard and map it on to a new I/O address space, available on the I/O channel and map the keyboard controller on to this area, using the hardware mapping technique Thus both tasks will be having separate keyboards

REFERENCES

1 Nelson, Ross P , The 80386 / 80486 Programming Guide , Microsoft Press, USA, (1991)

2 PC / AT Technical Reference, IBM Corp, (1984)

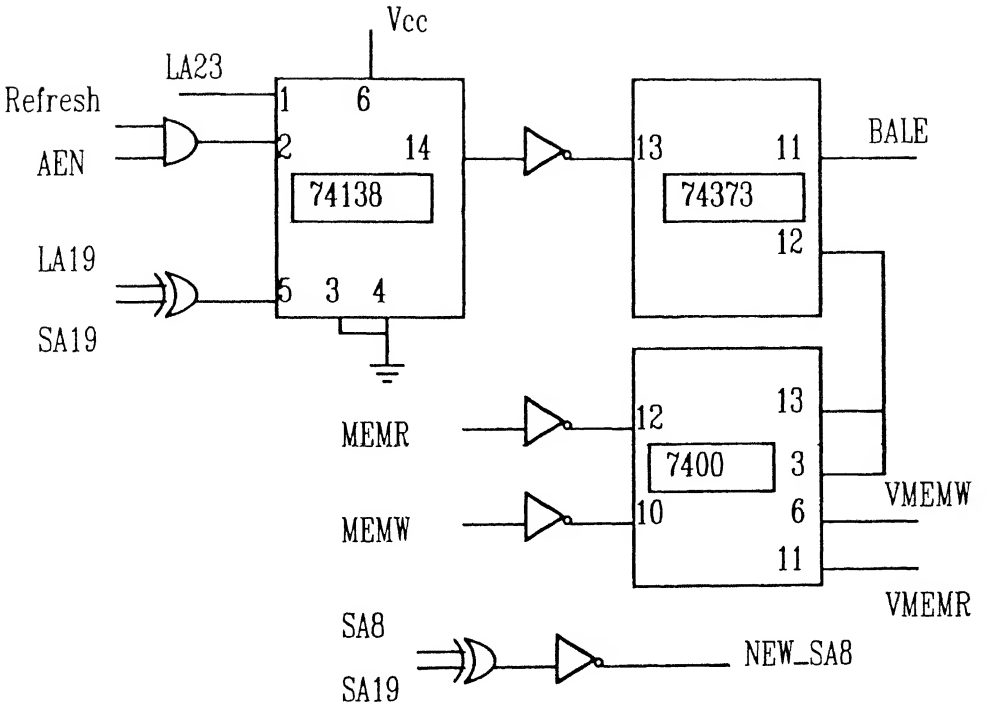
3 Norton, Peter, The PC Programmer's Bible , Microsoft Press, USA, (1993)

4 Pappas, Chris & Murray, William, 80386 Handbook , Osbourne McGraw Hill, USA, (1988)

5 Duncan, Ray, Encyclopedia DOS , Microsoft Press, USA, (1990)

6 TTL Data Hand - Book, Galgotia Publications, (1988)

APPENDIX



Display Card Mapping Circuit Diagram

IME-1995-M-SOM-CPU